INFORMATION AND CONTROL 29, 162–184 (1975)

Web Automata*

DAVID L. MILGRAM

Department of Mathematics, University of Maryland, Baltimore, Maryland 21228

A language is commonly defined to be the set of all input "tapes" which cause an automaton eventually to enter a final or "accepting" state. This concept has been generalized to allow labeled graphs or "webs" to be input to a "web" automaton—a type of automaton which can negotiate its way about the input web by labeling and relabeling the edges and/or nodes of the web. A web language is thus the set of webs for which a web automaton eventually enters a final state. The web languages defined in this way are precisely those generated by web grammars.

"Node" and "Edge" Automata are web automata which may use only node and edge labels, respectively, during their intermediate computations. General web traversal algorithms are exhibited and proved for both types of automata. Several possible web analogs to the linear bounded automaton (LBA) are considered. The node automaton versions are shown to be weaker than the edge automaton versions. Other strict containment and equivalence relationships among these automata are presented.

1. INTRODUCTION

Working in the area of scene analysis, one eventually realizes that a two-dimensional picture has meaning which goes beyond the power of two-dimensional representation. The combination of picture segments into identifications of higher order picture parts requires a multidimensional relational structure. The labeled graph or "web" is only one of a large variety of formal objects which have been created to provide for the storage and manipulation of descriptions of pictorial information. There is no a priori reason to believe that webs have greater descriptional power than multigraphs or graphs of bounded degree. However, webs do provide a simple yet general formalism which invites theoretical speculation.

* This work was done while the author was at the Computer Science Center, University of Maryland, College Park, Maryland 20742. The support of the Office of Computing Activities, National Science Foundation, under Grant GJ32258 is gratefully acknowledged, as is the help of Eleanor B. Waters and Shelly Rowe in preparing this paper.

In speculating about webs, various questions come to mind: Does it make any difference whether we label the edges or the nodes, or both? Is there a relationship between the automorphism group of a web and its storage capacity? Are some classes of webs inherently better storage media for relational information than others? These and other similar questions can be formalized within automata theory. It was felt that the definition and exploration of web automata properties could provide answers to these questions. In this paper we provide such a definition and investigate the properties of web automata, from the point of view of automata theory.

2. Webs and Web Automata

2.1. *Webs*

Let N be a denumerable set of *node identifiers*, e.g., the positive integers. Let L_N, L_E be finite sets of *node labels* and *edge labels*, respectively, each with a distinguished element $\#$. A *generalized directed web* ω over (L_N, L_E) is a pair (f, g) of total functions, where

$$f: N \to L_N \text{ and } N_\omega = \{n \mid f(n) \neq \#\} \text{ is finite}$$
$$g: N \times N \to L_E \text{ and } E_\omega = \{(n_1, n_2) \mid g(n_1, n_2) \neq \#\} \text{ is finite.}$$

We shall assume ω to be loop-free, i.e., $\text{diag}(N \times N) \cap E_\omega = \varnothing$.

A *generalized undirected web* (*GUW*) is a generalized directed web (f, g) in which g is symmetric. We shall assume from now on, unless expressly stated otherwise, that all webs are undirected, and shall regard all pairs (n_1, n_2) as unordered.

Let $M_\omega = \{n_1 \mid (n_1, n_2) \in E_\omega \text{ for some } n_2\}$. Thus M_ω is the set of nodes at the ends of non-$\#$ edges. Clearly M_ω is finite. A *GUW* ω will be called *consistent* iff $M_\omega \subseteq N_\omega$, i.e., if every such node has a label other than $\#$. By the underlying graph of a web ω, we mean the graph (N_ω, E_ω).

Let $E_\omega^{\,2} = \{(n_1, n_3) \mid \exists n_2; (n_1, n_2), (n_2, n_3) \in E_\omega\}$. $E_\omega^{\,k}$ and $E_\omega^{\,\infty}$ are defined similarly. Thus $E_\omega^{\,\infty}$ is the set of pairs of nodes that are joined by paths consisting of non-$\#$ edges.

Let $D_\omega = E_\omega \cap (N_\omega \times N_\omega)$. Thus D_ω is the set of non-$\#$ edges both of whose nodes are non-$\#$. Note that if ω is consistent we have $D_\omega = E_\omega$. A *GUW* ω will be called *node-connected* iff $N_\omega \times N_\omega \subseteq D_\omega^{\,\infty}$. Similarly, ω will be called *edge-connected* iff $M_\omega \times M_\omega \subseteq E_\omega^{\,\infty}$.

Examples. (a) $A \text{—}^a \# \text{—}^b B$ and $A \text{—}^a B \text{—}^\# C$ are edge-connected but not node-connected.

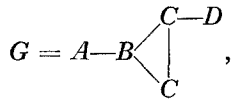
(b) $A \text{—}^a B \text{—}^\# \# \text{—}^b \#$ is node-connected but not edge-connected.

PROPOSITION 1. *If ω is consistent and node-connected, it is also edge-connected.*

Proof. $M_\omega \times M_\omega \subseteq N_\omega \times N_\omega \subseteq D_\omega^\infty = E_\omega^\infty$. ∎

We shall assume from now on that all webs are consistent and connected. This condition could in any case be achieved by suitably modifying the definitions of our automata.

We shall occasionally talk about labels on a diagram representing a web as if they were identifiers. This will only be done when no ambiguity can result. For example, if

$G =$ 

,

we may talk about the "node" B or the "edge" (C, D), but not the "edge" (B, C).

2.2. *Web Automata*

A *generalized web automaton* (WA) is a 7-tuple

$$A = (\Sigma, L_N, L_E, \varphi_+, \varphi_-, s_0, F),$$

where

Σ is a finite nonempty set of *states*,

L_N is a finite nonempty set of *node labels* with # distinguished,

L_E is a finite nonempty set of *edge labels* with # distinguished,

$s_0 \in \Sigma$ is the *start state*,

$F \subseteq \Sigma$ is the set of *final states*, and

$\varphi_\pm: \Sigma \times L_E \times L_E \times L_N \times L_N \to 2^{\Sigma \times L_E \times L_N}$ are the *transition functions*.

An *instantaneous description* (ID) of a WA on the web $\omega = (f, g)$ is a triple $((f, g), (n_1, n_2), s)$ where $n_1, n_2 \in N$, $s \in \Sigma$. The interpretation of φ is as follows: For the ID above, let $f(n_2) = p_1$ and $g(n_1, n_2) = q_1$.

(a) Suppose $\varphi_+(s, q_1, q_2, p_1, p_2) \ni (s', q', p')$. If for some n_3 we have $g(n_2, n_3) = q_2$ and $f(n_3) = p_2$, then let f' be the same as f except that $f'(n_2) = p'$ (we write this as $f' \equiv f|_{f'(n_2)=p'}$). Also, $g' \equiv g|_{g'(n_1,n_2)=q'}$. The new ID is $((f', g'), (n_2, n_3), s')$.

(b) Suppose $\varphi_-(s, q_1, q_2, p_1, p_2) \ni (s', q', p')$. If for all n_3 it is the case that $g(n_2, n_3) = q_2$ implies $f(n_3) \neq p_2$, then let

$$f' \equiv f|_{f'(n_2)=p'}, \qquad g' \equiv g|_{g'(n_1,n_2)=q'};$$

the new *ID* is $((f', g'), (n_1, n_2), s')$.

An informal description of the above transition functions has the web automaton in some state with a current position along a labeled edge facing a labeled node, called the "tip" node. Both of these labels are known to the automaton. In addition, for any edge label e and any node label a, the automaton can ask if there is an e-labeled edge with tip node labeled a adjacent to the current edge. If there is such an adjacent edge-node pair, the φ_+ function allows the automaton to relabel the current edge-node pair, to move to the adjacent edge-node position, and to change its state. If no such edge-node pair exist adjacent to the current position, the φ_- function detects it and allows the automaton to relabel the current edge-node pair and to change state. No change in position accompanies the φ_- transition. Note that in order to enlarge the graph the automaton must step off the marked portion of the graph. Also note that the automaton can reverse direction on an edge by using the φ_+ specifying an edge-node label pair corresponding to the labels on the current edge and the node opposite the tip node along that edge.

The transition functions in the definitions above depend on the labels that are present in the neighborhood of the automaton's position. This requires some further explanation. For automata on tapes (or even on n-dimensional tapes, where n is fixed), the neighborhood of the automaton's position always has bounded size (2, for tapes), and the neighboring positions are always distinguishable from one another since they lie in different directions (left and right). Thus such an automaton can systematically explore its neighborhood, by moving in each direction, and determine whether specified labels are present or absent, in a bounded number of moves. In a web, however, directions are not distinguished (except for "in" and "out" on a directed web; but there can be an unbounded number of each at any position). If a *WA* attempts to explore its neighborhood by moving to some arbitrary neighboring position, it cannot find its way back to the starting point, which in general is only one of many neighbors of the new position. In order to permit systematic exploration, the *WA* must be given the power to sense the neighboring labels before it moves. It can then explore systematically by giving its current position a unique label, moving to a neighbor and marking it, moving back to the uniquely labeled neighbor, moving to an unmarked neighbor, moving

back to the uniquely marked neighbor, and so on. When no unmarked neighbors remain, the exploration is complete. The marks can be erased by reversing the process. Note that in order for the exploration to terminate, the *WA* must be able to sense that no unmarked neighbors remain. The function φ_- provides this capability. One cannot test for the absence of a symbol from a neighborhood by verifying that other symbols are present in all the neighboring positions since the number of such positions is not a priori bounded. Thus a *WA* must be able to sense "negative context" as well as "positive context". To emphasize the distinction between the two types of context, two distinct transition functions φ_+ and φ_- have been used, rather than a single function defined for two disjoint sets of "presence-testing" and "absence-testing" states.

We define the *language* L of a *WA* as the set of initial webs for which the *WA* ever enters a final state, when started on the web in the initial state. When we are interested in the language of a *WA*, we will call the *WA* an *acceptor*. In general, the initial webs will be defined with respect to subsets L_N' and L_E' of L_N, L_E. We will say that the language L is *over* (L_N', L_E').

As discussed in Section 2.1 we want all webs to be consistent and connected. This can be guaranteed by requiring that the input web be consistent and connected and inserting a "preprocessor" which traverses the web before making a proposed move and allows the transition to proceed only if the web would remain consistent and connected. Traversal and similar algorithms are discussed in Section 3.

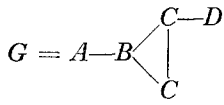
Since webs in general have no uniquely identifiable points, unlike tapes and arrays (leftmost point, leftmost point on the topmost row), we cannot define acceptance of a language from some initial point. If we provide the web with a uniquely labeled edge or node, then of course we can investigate web automata started at this unique position. However, since the position of the label would be arbitrary, nothing is to be gained in this way; the same language is also defined without the use of a unique label.

It should be noted at this point that a *WA* may be simulated by a Turing acceptor since any web ω of order n can be represented by an $n \times n$ incidence matrix, in which the matrix elements represent the edge labels, together with an extra row of length n which represents the node labels. This $n + 1 \times n$ matrix can now be represented as a Turing tape of length $n^2 + 1$.

2.3. *Determinism*

An important point to be made here concerns the notion of determinism. Most types of automata are called *deterministic* if the value of the transition function for any argument for which it is defined is a singleton. This concept

applies to web automata; its failure to hold may be called "compile-time" nondeterminism since it can be detected at the definitional level. Even if it does hold, some 5-tuple still may apply at any number of different positions in the neighborhood of the current position. For example, if

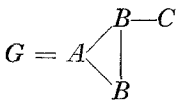


and if the automaton's current position is on (A, B) and it can move to some (B, C), then it may move to either of the C's, there being no way to distinguish one from the other at this transition. (If D were absent, then the C's would be equivalent up to a graph automorphism, and it would make no difference which C the automaton chose.)

This type of nondeterminism may be called "execution-time" nondeterminism since its occurrence is determined by "running" the automaton on a web.

PROPOSITION 2. *Compile-time nondeterminism is a decidable property of web automata; execution-time nondeterminism is not decidable.*

Proof. It is clear that the former may be decided by simply examining $\varphi_{\pm}$. To prove the latter undecidable, let T be any deterministic Turing acceptor. Let W_T be a (directed) web acceptor constructed to simulate T and which accepts the same language. T is deterministic, hence W_T is both compile-time and execution-time deterministic. Now consider the set of webs consisting of strings with the graph



appended to one end (A, B, C being symbols not appearing elsewhere.) Call such a web "augmented". Finally devise a web acceptor W_T' which imitates W_T, treating A as an end marker, provided that if W_T accepts the unaugmented string, then W_T' moves to the right end of the string and onto G, verifying that it is the correct G. Then W_T' is execution-time nondeterministic iff $L(W_T)$ is nonempty, i.e., iff $L(T)$ is nonempty. Since the nonemptiness problem for Turing acceptors is undecidable, we have the stated result. ∎

2.4. *Other Web Automaton Formalisms*

The technical literature contains numerous attempts to define automaton-like devices on graph structures. Two essential features of our definition of web differentiate these automata from a *WA*. We place no a priori bound on the degree of the *WA*'s input graph. Also, we assume no given ordering of the edges at a node. The graph property recognition machines of (Shank, 1971a) and the graph automata of (Mylopoulos, 1972), on the other hand, provide a priori edge orderings. (The former correspond in power to Isotonic Edge Automata; the latter to Isotonic Node Automata with an extra memory tape. For the definitions see Sections 3 and 4.)

Graph automata formalisms in the area of program networks have been proposed by (Fisher and Raney, 1969) and (Rutledge, 1970). Their interest lies in the sets of strings generated by "tours" of the graph corresponding to the ordered string of symbols at the nodes visited by the automaton.

Two models of parallel graph automata have been proposed: (Rosenstiehl *et al.*, 1972) and (Shah *et al.*, 1973). Rosenstiehl's "intelligent graphs" are networks of simple automata having a given fixed degree, with each automaton able to distinguish among its neighbors. This restriction provides a rich setting in which to describe various algorithms. When this restriction is relaxed, as in the Shah definition, the natural symmetries in the graph limit even the simplest recognition powers of the parallel automaton.

For computations of Turing machines expressed as graph automata (see Savitch, 1972; Shank, 1971b).

2.5. *Web Grammars*

Among the grammatical formalisms employing graph constructs are the web grammars of Rosenfeld and Milgram (1972), which have been proved equivalent to Node Automata. A simple generalization of their definition can be shown equivalent to the Generalized Web Automaton notion. Other web formalisms have been proposed by Pfaltz and Rosenfeld (1969, 1972), Montanari (1970), Abe *et al.*, (1973), Mylopoulos (1972), and Schneider (1970). Context-free graph grammar models have been defined by Pratt (1971), Feder (1971), and Pavlidis (1972). A category–theoretic approach to graph grammars has been taken by Ehrig *et al.*, (1973). Graph grammars designed for specific application areas will not be reviewed here. Discussion of several may be found in Earley (1971), Rosenfeld and Strong (1971), Schwebel (1972), and Shaw (1972).

3. Register, Node, and Edge Automata

In this section we investigate the types of automata which arise when restrictions are placed on the node or edge label sets (L_N and L_E in the *WA* 7-tuple).

The set of input webs to a *WA* is ordinarily the set of all webs over the (node, edge) label sets (L_N', L_E') where $L_N' \subseteq L_N$ and $L_E' \subseteq L_E$. Thus to compare the powers of different types of automata, we must choose input webs which are defined over the intersections of the input web label sets of the automata. At the very least, we would want to be able to talk about automata which accept ordinary graphs and for this a single non-# edge label in L_E' suffices. The set of graphs over $(\{\#, a\}, \{\#, e\})$ provides a consistent set of input webs to test the power of various types of restricted *WA*'s. For completeness, though, we begin by considering a more elementary case with a consequently more elementary input web structure.

3.1. $L_E = \{\#\}$—*Register automata*

A *WA* A whose edge label set consists of the single background symbol # will be called a "*register*" *automaton* (*RA*). Let $L_N = \{\# = a_0, a_1, \ldots, a_n\}$. Since there is no notion of adjacency, a web ω may be described simply by an n-tuple $(m_1, \ldots, m_n)$ where m_i, $i = 1, \ldots, n$, is a nonnegative integer whose value is the number of occurrences of the symbol a_i on the nodes of ω. Such an ensemble may be thought of as a "bag" of symbols. Each m_i is potentially unbounded and thus the storage structure of a bag is identical to that of an n-register program machine. Each position in the n-tuple will be called a *register*.

Let A be an *RA*. An *instantaneous description* of A on ω is a triple $((m_1, \ldots, m_n), i, s)$ where $(m_1, \ldots, m_n)$ is the n-tuple describing ω, $i \in \{0, \ldots, n\}$ is the register whose contents are being scanned, and s is the *current state* of A.

Note that m_0 has not been defined since, according to our definition of web, there are always an infinite number of #'s in the background. Alternatively, we could simply consider m_0 to be identically 1. Using φ_+, the automaton can pass from one register to another, in the process, incrementing or decrementing its contents by 1. Using φ_-, the automaton can determine whether a register is zero and change state appropriately. Thus a register automaton has the familiar powers of a program machine (Minsky, 1967). Minsky states that computability by program machine with two registers is equivalent to computability by a Turing machine if the input tape to the Turing machine is properly encoded in one of the registers. We thus have the following proposition.

PROPOSITION 3. *Let T be any Turing machine and α an input tape consisting of 1's; then there is a 2-register automaton A_T which can simulate T on α when started in the ID* $((2^{|\alpha|}, 0), 1, s_0)$.

Note that there is a 3-register automaton A_T' which can simulate T on α when started in the *ID* $((|\alpha|, 0, 0), 1, s_0)$. The extra register is needed to implement a preprocessing step which changes the web $(|\alpha|, 0, 0)$ to $(2^{|\alpha|}, 0, 0)$ by repeatedly doubling the size of a register initialized to 1 while decrementing the $|\alpha|$ register to zero.

The *language accepted* by a register automaton is the set of input webs for each of which the automaton ever enters a final state. If $L_N' = \{\#, a_1\}$, each input web is characterized by the integer m_1 and thus the language of a register automaton is a subset of the nonnegative integers. The underlying graph structure is, from the point of view of web automata, uninteresting. Nonetheless, because of their unbounded storage capacity, *RA*'s can simulate *WA*'s for the set of input webs over $(L_N', \{\#\})$. *WA*'s, of course, can create and rewrite edges while register machines cannot.

THEOREM 1. *Let A be any WA and let W be the set of webs over $(L_N', \{\#\})$ accepted by A. Then there exists RA A' which accepts W.*

Proof. A' operates by simulating a Turing machine simulating the *WA* A. This is seen to be possible once the webs over (L_N, L_E) have been assigned Gödel numbers. The Gödel numbering of a web $\omega = (N_\omega, E_\omega, (f, g))$ is accomplished by first assigning Gödel numbers to each of the finite sets, N_ω and E_ω. We can represent the functions f and g by considering the sets of pairs $\{(n, f(n)) \mid f(n) \neq \#\}$ and $\{((n_1, n_2), g(n_1, n_2)) \mid g(n_1, n_2) \neq \#\}$. Each of these may be assigned a Gödel number since each is a finite set. The Gödel number of the web w is now derived from the Gödel numbers of N_ω, E_ω, f, and g. The relabeling operations of A then become arithmetic functions mapping Gödel numbers into Gödel numbers and can be performed by a Turing machine and (by Proposition 3) simulated by A'. ∎

3.2. $L_E = \{\#, e\}$—*Node automata*

The addition of a non-# symbol to the edge label set allows us to represent an arbitrary graph structure by using e as the label for edges in the graph. A *WA* restricted to this edge label set is called a "node automaton" (*NA*). While edges can be added or deleted, the primary information storage capacity of the underlying graph resides in the nodes. Like the *WA*, the node automaton has the power to traverse an arbitrary web and return to its starting point without modifying the graph or any edge labels. This algorithm is exhibited and its validity proved in Appendix 2.

THEOREM 2. *Let W be the set of webs over $(L_N', \{\#, e\})$ accepted by some WA A. Then there exists an NA A' which accepts W.*

Proof. Let $L_E = \{\#, e, a_1, \ldots, a_k\}$ be the set of edge labels of A and L_N the set of node labels of A. Then A' will have a node label set containing distinct symbols for each symbol in L_N and L_E-$\{\#\}$. A' traverses the initial web and between every pair of nodes joined by an e-labeled edge, A' inserts a new node labeled with the node symbol e. [The node insertion is done by first marking the particular endpoints with unique labels, finding a node not on the graph (i.e., picking a node with label $\#$ that is not on any edge with non-$\#$ label), joining each of the endpoints to the new node along e-labeled edges, destroying the edge between the two marked endpoints, marking the new point with node symbol e, and erasing all marks local to this process.] Thus A' simulates the operation of A using an extra node to "remember" the edge label that A is using. Subsequent insertions or deletions (i.e., changes of edge labels from $\#$ to non-$\#$, or vice versa, by A) are handled by A' in a similar manner. If A ever enters a final state, then so does A'. This completes our sketch of the proof of Theorem 2. ∎

3.3. $L_N = \{\#, a\}$—*Edge Automata*

By restricting the node label set of a WA to the doubleton $\{\#, a\}$, and maintaining consistency, one defines the class of "edge" automata (EA), i.e., automata using only edge labels. We have immediately for such edge automata.

THEOREM 3. *Let L be the set of webs over $(\{\#, a\}, L_E)$ accepted by a given WA A. Then there exists an EA A' such that $L(A') = L$.*

Proof. Let P be the node label set and Q the edge label set of A; we assume $P \cap Q = \varnothing$. A' will have an edge label set with a subset corresponding to $P \cup Q$. A' begins by adding a new node, the "star" node, to the initial web and connecting it to every node of the initial web. All new edges thus added are identifiable by a unique star label. Whenever A would label some node with a symbol from P, A' labels the edge from that node to the star node with the corresponding edge label. Thus the labeled edge from every node to the star node "remembers" the node label, so that A' simulates A and $L(A') = L(A)$. ∎

As in Theorem 2, the proof of this theorem requires a traversal algorithm. An edge traversal algorithm is established in Appendix 1.

In each of the proofs of Theorems 1–3, the restricted automaton simulating the WA created additional storage to contain the extra symbols that the WA

was using. The amount of such additional storage gives some indication of the relative power of each restricted type of automaton vis-a-vis the *WA*. The 2-register automaton of Proposition 3 uses exponentially more space than a Turing machine. In fact, no matter how many registers are available, a register automaton cannot simulate a Turing machine in less than exponential storage. To see this, note that with linear storage of size n, a Turing machine can have order 2^n instantaneous descriptions. A k-register automaton in which each register is bounded in size by $f(n)$ has order $f(n)^k$ *ID*'s. Thus for bounded k, f must be exponential.

In Theorem 2, the node automaton introduces a new node (and a new edge) for each edge of the (simulated) web. Inasmuch as a graph with n nodes may have up to $[n(n-1)]/2$ edges, the new nodes may account for quadratic growth in the size of the web. On the other hand, the edge automaton in Theorem 3 need only introduce one extra node (while doubling the number of edges). In Section 4, the edge automaton result will be improved so that no change to the underlying graph need be made to accommodate the extra node information. This is clearly a best possible result insofar as storage efficiency is concerned. One may therefore ask whether the result for node automata may be similarly improved, or, alternatively, whether the quadratic growth in Theorem 2 is essential. This, too, will be resolved in Section 4 where it is proved that node automata are inherently less powerful than edge automata in the storage capacity sense.

4. Web-Bounded Automata

The relationship between computational storage capacity and language complexity has been an area of continuing interest in automata theory. We consider various restrictions which define automata types several of which correspond to linear bounded automata in the one-dimensional tape case.

In what follows, we consider only sets of input webs over $L_N = \{\#, a\}$, $L_E = \{\#, e\}$, thus allowing a uniform comparison of node, edge, and generalized web automata. The simplest way of defining a web-boundedness restriction is to say that the underlying graph structure of the input web is unchanged by the computation. This corresponds in the two-dimensional case to isotonic array automata.

Definition. An *isotonic web automaton* (*IWA*) is a *WA* whose φ_+ and φ_- functions never rewrite a #-edge with a non-# label and vice versa.

An *isotonic edge* (*node*) *automaton*, called an *IEA* (*INA*), is an *IWA* which is also an edge (node) automaton.

We also will consider the following relaxations of the isotonic restriction.

DEFINITION. An *erasing web automaton* (*EWA*) is a *WA* whose φ_+ and φ_- functions never relabel a #-edge with a non-# symbol.

An *erasing edge* (*node*) *automaton*, called an *EEA* (*ENA*), is defined similarly. These automata are called "erasing" because they are allowed to erase edges, i.e., to relabel non-# edges with #. It is assumed, of course, that the underlying graph does not become disconnected during any erasure.

Remark. Both isotonic automata and erasing automata reduce to the familiar class of linear bounded automata when restricted to operating on one dimensional tapes.

Lastly, we consider a formalism which allows the underlying graph to increase in size only moderately; by adding edges but not new nodes, the number of edges may be increased until the underlying graph is a complete graph.

DEFINITION. A (*node-*) *bounded web automaton* (*BWA*) is a *WA* whose φ_+ and φ_- functions never relabel a #-node with a non-# symbol and which guarantees that M_ω is nonincreasing for the underlying graph ω.

A *bounded edge* (*node*) *automaton*, called a *BEA* (*BNA*), is defined similarly.

A *BEA* whose input web is a one-dimensional tape of length N has $N - 1$ edges in its storage space arranged in a linear fashion. By adding edges between the existing nodes, a storage space with $[N \cdot (N - 1)]/2$ edges may be created, although this set of edges no longer has a natural order of access. It seems reasonable to conclude that this quadratic increase in storage space should enhance the power of *BEA*'s beyond that of *EEA*'s or *IEA*'s. The similar question for *BNA*'s, though, is not quite so obvious. In the rest of this chapter, we investigate the relationships between the classes of automata (as acceptors) which have been just defined.

The following table clarifies the various distinctions between the classes of restricted automata.

	WA (Node and edge data storage)	*EA* (Edge data storage)	*NA* (Node data storage)
isotonic (underlying graph is constant)	*IWA*	*IEA*	*INA*
erasing (underlying graph is nonincreasing)	*EWA*	*EEA*	*ENA*
bounded (underlying graph's node set is nonincreasing)	*BWA*	*BEA*	*BNA*

Let XYA be some automaton type of those listed above. By $\mathscr{L}_{XYA}$ is meant the class of those sets of webs over $(\{\#, a\}, \{\#, e\})$ accepted by XYA's.

Scanning the table vertically, we have immediately that $\mathscr{L}_{IYA} \subseteq \mathscr{L}_{EYA} \subseteq \mathscr{L}_{BYA}$, for $Y \in \{W, E, N\}$. The following theorem shows that two of these containments are equalities.

THEOREM 4. $\mathscr{L}_{IWA} = \mathscr{L}_{EWA}$, $\mathscr{L}_{IEA} = \mathscr{L}_{EEA}$.

Proof. We prove the first of these equalities. The second follows similarly. Let A be an EWA; let $\$_E$, $\$_N$ be new symbols not in L_E, L_N, respectively. Let A' be like A except that whenever A deletes a node (by relabeling it as $\#$), A' labels it with $\$_N$, and similarly for deleting edges. A' reads $\$_N$ as if it were $\#$, and similarly for $\$_E$. Thus $L(A) = L(A')$. ∎

This proof does not apply to ENA's since for an ENA, $L_E = \{\#, e\}$ and additional edge labels are not permitted. In general, it does not seem possible to represent the presence of an edge between any two nodes by node labels alone. This appears to be true even for webs of bounded degree. We thus have the strict containment for node automata.

THEOREM 5. $\mathscr{L}_{INA} \subsetneqq \mathscr{L}_{ENA}$.

Proof. We will construct an erasing node automaton language which is accepted by no isotonic node automaton. We will use the following lemmata. K_N is the complete graph of order N.

LEMMA. *Let $K = \{K_N \mid N \geqslant 1\}$. Let A be an INA and let $K_A = L(A) \cap K$. Then there is an INA A' such that $L(A') = K_A$.*

Proof. A' operates by first determining whether the input graph is complete, i.e., for each node, A' determines that every other node is a neighbor. Upon its successful determination that the input graph is complete, A' begins the simulation of A. If A' ever accepts a graph g then g is already in K and thus $g \in K_A$. ∎

DEFINITION. Let $N(K_A) = \{N \mid K_N \in K_A\}$, for example, $N(K) = Z^+$.

DEFINITION. If ω is the graph (N, E), then $\bar{\omega}$, the complement of ω, is the graph $(N, N \times N - E - (N, N))$. Thus $\bar{\omega}$ has an edge between any two nodes not joined in ω. Note that $\bar{\omega}$ need not be connected. Also $\omega = \bar{\bar{\omega}}$.

LEMMA. *Let W be the language of some INA A; then there exists an INA A' whose language is W', the set of connected graphs whose complements are in W.*

Proof. Note that for any sentential form ω derived from input g, the underlying graph is always g, by definition of *INA*. Suppose at some stage that A is in state S, scanning a node labeled a and testing the existence of a neighbor labeled b, i.e., deciding whether $\varphi_+(s, a, b)$ applies. If the simulated state is s and A' is currently scanning a node labeled a, then A' simulates this by relabeling the current node with a special symbol a', then traversing the graph and marking all b's not adjacent to a' with a special symbol. A' then nondeterministically chooses one of the specially marked b's, performs the indicated φ_+ transition, erases all temporarily marked symbols, and moves to its new position.

A φ_- move is simulated in a like manner except that the traversal step uncovers no b's not adjacent to a's. In that case, A' returns to a' and performs the φ_- transition. Since ω' contains only connected graphs, the traversal will be effective in determining the presence or absence of b's. ∎

LEMMA. *Let X be a subset of Z^+. Then $\{K_N \mid N \in X\} = K_X$ is accepted by some INA A iff $\{1^{|x|} \mid x \in X\}$ is accepted by some Turing acceptor T with log $|\ x\ |$ (read/write) storage.*

Proof. In the discussion of register automata in Section 3, it was noted that an *ID* is a triple $((m_1, \ldots, m_n), i, s)$. Although we have not included it in the definitions above, an isotonic register automaton is clearly a register automaton which never relabels #-nodes. Thus the total amount of register space is just $|\ \omega\ |$, the size of the input web. Note that the complement of the input web ω of a register automaton is $K_{|\omega|}$, the complete graph on $|\ \omega\ |$ nodes.

We have therefore by the previous Lemma that isotonic node automata on complete graphs are equivalent to isotonic register automata. Thus an *ID* for an *INA* A on a complete graph ω has the form of the triple $((m_1, \ldots, m_n), i, s)$ above. The number of different *ID*'s for a given *INA* A on ω is clearly bounded by $|\ \omega\ |^N \cdot |\ \omega\ | \cdot |\ A\ |$ where $|\ A\ |$ is the size of A's state set. Thus any *ID* can be represented (in binary) in

$$\log(|\ \omega\ |^N \cdot |\ \omega\ | \cdot |\ A\ |) = (N + 1) \log |\ \omega\ | + c \text{ storage.}$$

Since N is a constant once we are given A, we may use standard data compression techniques to reduce this to $\log |\ \omega\ |$ storage. We can now construct a Turing acceptor T which can create the initial *ID* and then transform the current *ID* into the new *ID*. This transformation is achieved by subtracting 1 from one register, adding 1 to another, and modifying the *RA*'s position and state. Thus T accepts $1^{|x|}$ whenever A accepts K_X.

Conversely, let X be the language over $\{1\}$ of some Turing acceptor T using $\log N$ storage. Suppose T has q states and a working alphabet of size M. An *ID* of T consists of the position of the read head on the input tape, the contents of the auxiliary tape under and to the right of the read/write head on the auxiliary tape, the contents to the left, and T's state. We can thus represent an *ID* of T with one unary register of length $|x|$, and two m-ary registers of maximum size $\log |x|$. An m-ary register of size $\log |x|$ is equivalent to $\log m$ unary registers of size $|x|$. Thus a register machine with $2 \cdot \log m + 1$ registers can simulate T. By the previous Lemma, there is an *INA* which simulates T. ∎

Finally, we quote a theorem by Ibarra (1972).

THEOREM (Ibarra). *Let $L_1(n), L_2(n)$ be fully constructable tape functions with $L_2(n) \geqslant [\log_2 n]$. Then there is a language in $\mathscr{L}_N(L_1(n))$ which is not in $\mathscr{L}_N(L_2(n))$, providing the following conditions are satisfied:*

(a) *There exist $k \geqslant 2$ and tape functions $f_1(n) \geqslant n, \ldots, f_k(n) \geqslant n$ such that $L_2(f_i(n))$ is fully constructable for $1 \leqslant i \leqslant k$.*

(b) $\mathscr{L}_N(L_2(f_{i+1}(n))) \subseteq \mathscr{L}_N(L_1(f_i(n)))$ *for* $1 \leqslant i \leqslant k-1$.

(c) $L_1(f_k(n)) \geqslant [\log_2 n]$ *is fully constructable and*

$$\inf_{n\to\infty} ((L_2(f_1(n)))^2 / L_1(f_k(n))) = 0.$$

Let $L_2(n) = \log_2 n$ and $L_1(n) = n$ and letting $k = 2$, $f_1(n) = f_2(n) = n$ we have satisfied conditions a, b, and c. Thus there exists a set P of unary numbers accepted by some *LBA*, T, which is accepted by no *TA* whose auxiliary storage is bounded by the log of the length of the input.

We now conclude the proof of Theorem 5. Let $C_p = \{K_N \mid N \in P\}$. We have established that C_p is not accepted by any *INA*. We will construct an *ENA* T' which accepts C_p. T' first verifies that the input web is complete. It then deletes $N-2$ edges at its current position and marks the current node as "finished." It goes to an unfinished node and deletes $N-3$ edges (never deleting an edge bounded by a finished node). Continuing in this manner, recursively, T' changes the underlying graph of the input web into a tape of length N. T' now imitates T on this tape. Thus T' accepts C_p. ∎

DEFINITION. The *line graph* of a graph g, Line (g), is the graph obtained by creating a node for each edge of g and joining together just those nodes corresponding to edges adjacent in g.

Remark. Every *IEA* on a set of webs $\{g\}$ can be construed as an *INA* on the set of line graphs $\{\text{Line}(g)\}$.

THEOREM 6. $\mathscr{L}_{EYA} \not\subseteq \mathscr{L}_{BYA}$ *for* $Y \in \{W, E, N\}$.

Proof. Let $ST = \{K_{1,N}\}$, the set of "stars." Let $ST_p = \{K_{1,N} \mid P(n)\}$ with P from Theorem 5. Note that $\text{Line}(K_{1,N}) = K_N$. It is immediate, therefore, that $ST_p \notin \mathscr{L}_{EEA}$. Furthermore, since all nodes except the star node are indistinguishable, we have $ST_p \notin \mathscr{L}_{ENA}$ and $ST_p \notin \mathscr{L}_{EWA}$. Let A be a *BEA* which adds suitably labeled edges to $K_{1,N}$ until the graph has become a "wheel"—all nodes are of degree 3 except the star node. The new edges constitute a cycle of length $N - 1$. By relabeling two adjacent edges as the beginning and end, respectively, A can now avail itself of a storage tape of length N. Thus A can be constructed to accept ST_p. This proves that $ST_p \in \mathscr{L}_{BEA}$ and $\mathscr{L}_{BWA}$. An almost identical argument applies for $\mathscr{L}_{BNA}$. ∎

Working now in the horizontal direction across the tableau, we have immediately that $\mathscr{L}_{XEA} \subseteq \mathscr{L}_{XWA}$; $\mathscr{L}_{XNA} \subseteq \mathscr{L}_{XWA}$; for $X \in \{I, E, B\}$. This is refined by the following theorem:

THEOREM 7. $\mathscr{L}_{XEA} = \mathscr{L}_{XWA}$, $X \in \{I, E, B\}$.

Proof. We need only show that $\mathscr{L}_{IEA} = \mathscr{L}_{IWA}$. Let A be any *IWA*; we construct an *IEA* A', which can simulate the use of node labels by A. Let A' have the edge label set of A plus the set of edge labels of the form (q, p, p') where q is an edge label of A and p, p' are any node labels of A. In addition, A' has states of the form (s, p) where s is a state of A and p is a node label of A. A' begins by traversing the web and rewriting all (marked) edges as $(q, \#, \#)$ where q was the original edge label. Upon completion of the traversal A' enters state $(s, \#)$. A' can now enter the simulation mode. Suppose a node currently labeled p is to be labeled p' by A, then we require that *all* marked edges at that node currently be of the form (q, p_1, p_2) where either p_1 or $p_2 = p$ (this is certainly true initially). A' relabels all those edges so that whichever of p_1 and p_2 was p is now p'. This is possible because whenever A' is on an edge (n_1, n_2) with label (q, p_1, p_2), it can determine whether n_1, n_2 have labels p_1, p_2 or p_2, p_1. [This is done in the following way: A' knows that initially it is on (i.e., pointing at) a node labeled $\#$. This is reflected in its initial state. Whenever A' is in a state (s, p) on and edge (q, p_1, p_2), then either $p = p_1$ or $p = p_2$ (or both). If $p = p_1$, then the node pointed at is labeled p_1 which means the other node on the edge must be labeled p_2.

Then when A' moves from (q, p, p_2) to (q', p_1', p_2') either $p = p_1'$ or $p = p_2'$ and its transition is to (s', p_2') if $p = p_1'$ or to (s', p_1') if $p = p_2'$.] By storing the node information on the edges A' can simulate A. ∎

COROLLARY. $\mathcal{L}_{XNA} \subseteq \mathcal{L}_{XEA}$, $X \in \{I, E, B\}$. *In fact, we have the stronger result*:

COROLLARY. $\mathcal{L}_{INA} \subsetneqq \mathcal{L}_{IEA}$.

Proof. $\mathcal{L}_{INA} \subsetneqq \mathcal{L}_{ENA}$ and $\mathcal{L}_{ENA} \subseteq \mathcal{L}_{EEA} = \mathcal{L}_{IEA}$ together imply $\mathcal{L}_{INA} \subsetneqq \mathcal{L}_{IEA}$. ∎

We have conjectured that $\mathcal{L}_{ENA} \subsetneqq \mathcal{L}_{EEA}$ but the question is still open. Similarly, $\mathcal{L}_{BNA} \subsetneqq \mathcal{L}_{BEA}$ is also conjectured true.

The established relationships between node and edge automata are indicated in the Hasse diagram below (Fig. 1). As is usual, solid lines indicate strict containment. Dotted lines indicate containments conjectured to be strict.

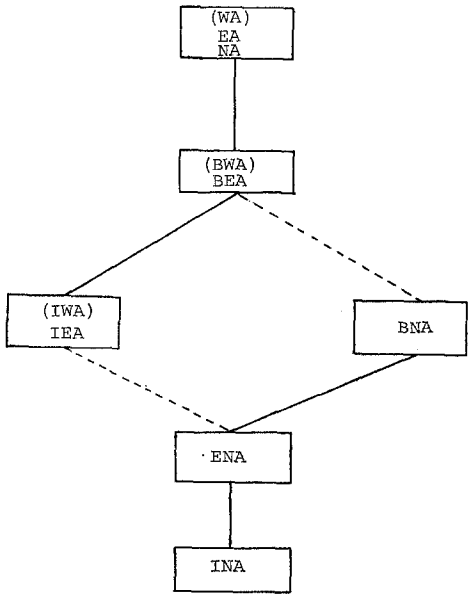


FIG. 1. Hierarchical relationships between edge and node automata.

5. Conclusion

In extending one-dimensional tape concepts to the web environment, we have encountered a variety of definitions corresponding to the linear bounded automaton. This fruitfulness signifies the need for further invetigation into the appropriateness of each definition. Different areas of application would quite likely prefer differing formulations and, in addition, would be content to constrain the classes of webs under discussion, e.g. planar graphs, cubic graphs, graphs of bounded degree.

As formal objects, web automata can be used to extend the theory of computational complexity to include the concept of computation with a read-only input tape and web-structured intermediate storage.

APPENDIX 1: Edge Traversal

In this Appendix we show that there exists an *IEA ET*, which can, from any initial position on an arbitrary web ω, visit all edges of ω and return to its initial position having erased all marks made in the course of the traversal.

Informally, *ET* operates as follows:

(1) Mark the initial edge with the symbol *a*.

(2) If there is no unmarked edge adjacent to the tip vertex of the current edge go to step (3). Otherwise move to the adjacent unmarked edge and do one of the following:

(2a) If there is an adjacent edge marked *a* or *c*, mark the current edge *b*, return to the position occupied at the previous entry to step (2), and reenter step (2).

(2b) If no adjacent edge is marked *a* or *c*, mark the current edge with an *a* and enter step (2).

(3) Reverse direction and do one of the following:

(3a) If there is no adjacent edge marked *a*, go to step (4).

(3b) Otherwise, mark the current edge with the symbol *c* and move to an adjacent edge marked *a*; then reverse direction and reenter step (2).

(4) If there is an adjacent unmarked edge, reenter step (2). Otherwise, mark the current edge with *c* and STOP.

We observe that the automaton's position upon any entry to step (2) is on an edge marked *a*.

PROPOSITION A1. *ET eventually stops.*

Proof. Unmarked edges get marked either *a* or *b* (step (2a) or (2b)); edges marked *a* get marked *c* (step (3b) or (4)). Eventually this marking process must stop. ▮

PROPOSITION A2. *Upon entry of ET into step* (2), *the edges marked a form a path. ET's initial position and current position* (*on entry to step* (2)) *are the two ends of the path.*

Proof. By algorithmic induction.

Basis: After step (1) the path consists of the initial edge marked *a*.

Induction Step: Suppose the proposition holds for the current entry into step (2); then we show that it holds at the next entry into step (2).

Indeed, if (2a) applies, then no change in the path has occurred. In (2b), a new edge is added to the path. This new edge is adjacent to *ET*'s previous position but to no other edge marked *a*. Thus the path has been lengthened by the new edge. If (3b) holds, *ET* moves back one edge from the end of the path and marks the abandoned edge *c*. Thus the path shrinks by one edge. Finally, (4) holds only if the path has shrunk to a single edge, and *ET* reverses its position on the single edge. ▮

COROLLARY. *ET always terminates in step* (4) *on the initial edge with no a-marked edges remaining.*

PROPOSITION A3. *When ET terminates, there are no unmarked edges left—that is, all edges are marked b or c.*

Proof. If not, there would remain some unmarked edge (n_1 , n_2) adjacent to a marked edge. Suppose the unmarked edge is adjacent to an edge labeled *c* and consider the edge (n_2 , n_3) last labeled *c* by *ET* adjacent to (n_1 , n_2). Just before *ET* labeled (n_2 , n_3) with *c*, *ET* had entered step (2) and was adjacent to (i.e., facing) the unmarked edge. [Otherwise, the unmarked edge would have been adjacent to the other end of (n_2 , n_3), and some edge in the path would be adjacent to (n_1 , n_2) after (n_2 , n_3) had been labeled *c*, which would contradict our assumption about (n_2 , n_3).] But edges get marked *c* by step (3b), which applies only if no unmarked edge is adjacent. Thus we have a contradiction. Suppose instead that (n_1 , n_2) is adjacent to an edge labeled *b*. Edges are labeled *b* iff they are adjacent at both ends to edges labeled *a* or *c*. Thus since all *a*'s are eventually changed to *c*'s, the *b* edge is adjacent at both ends

to c edges and the previous contradiction holds. Thus no such (n_1, n_2) exists. ▮

COROLLARY. *The edges marked c span the given web.*

PROPOSITION A4. *The edges marked c form a tree.*

Proof. By construction, every edge when labeled c was adjacent to a single a-marked edge. This defines a predecessor relation with the initial edge as root. ▮

We now show that *ET* can be modified to erase all b's and c's and return to its initial position. Let *ET* do the following instead of halting at step (4):

(1) Mark the initial edge d.

(2) If there is no adjacent c edge at the current tip vertex, enter step (2b); otherwise continue with (2a)

(2a) Move the adjacent edge, mark it e, and reenter step (2).

(2b) Erase all b's (if any) at the current tip vertex. Reverse direction and erase the e from the current edge. If there is an adjacent edge, move there, reverse direction and reenter (2). If there is no adjacent edge, move to the d edge and enter step (3).

(3) Same as (2), except that in (3b), if there is no adjacent e edge, move to the d edge, erase the d and stop.

Since the c's initially form a tree, the initial edge links two half-trees, each of which has its labels erased in turn. We have now an e-path similar to the a-path of *ET*. One end of the e-path is always adjacent to the d-edge. The d-edge cannot be traversed to the other half-tree until the e-path lying in the first half-tree is null. Thus all b's (each was adjacent to a c-edge) have been erased and all c's have been converted to e's and finally erased. The other half-tree is then erased and *ET* returns to the initial edge, in the initial orientation, erases the d and halts. ▮

APPENDIX 2: NODE TRAVERSAL

In this Appendix, an *INA* will be exhibited that from any starting point n on any web ω can visit all the nodes of ω and can then erase all the marks that it made in the course of this traversal and return to n. Informally, this "universal node traverser" *NT* operates as follows:

(1) Mark the initial node with the symbol u.

(2) If the current node (call it p) has an unmarked neighbor, mark p with x and go to such a neighbor (call it q). Otherwise, continue with step (3).

(2a) If q has a neighbor marked u, mark q with y, go to p (it is the unique neighbor of q marked x), mark p with u, and return to step (2).

(2b) If q has no neighbor marked u, mark q with z, go to p, erase all y marks from neighbors of p, mark p with u, go to q (it is the unique neighbor of p marked z), mark q with u, and return to step (2).

(3) If p has no unmarked neighbor, erase all y marks from neighbors of p and mark p with v.

(3a) If p has a neighbor marked u, go to such a neighbor and return to step (2).

(3b) If p has no neighbor marked u, STOP (i.e., enter a state signifying that traversal is complete; mark erasure will be discussed below).

The procedure defined by (1–3) will be referred to below as the Node Traversal algorithm (NT). It is understood that the marks u, v, x, y, z are distinct from the original node labels of ω, and that their presence does not destroy these labels.

PROPOSITION A5. *NT eventually halts.*

Proof. NT cannot loop through (2) alone without ever entering (3), since each pass through (2) marks an unmarked node (q) with either y or u, and the number of u's cannot increase indefinitely. But each time NT enters (3), it creates a v, and this too can only happen finitely many times. ∎

PROPOSITION A6. *At each entry of NT into step* (2), *the nodes marked u form a path $n_1, \ldots, n_k$ ($k \geqslant 1$) such that*

(a) *n_i is a neighbor of n_j in ω if and only if $j = i \pm 1$ (i.e., the path does not cross or touch itself).*

(b) *n_1 is the initial node, and n_k is the current node p.*

Proof. This is trivially true at the first entry into (2), since at step (1) the initial node was marked u and there are no other marks. Moreover, it is easy to see that if the proposition holds at a given entry into (2), it still holds at the next entry. Indeed, (2a) does not change the set of u's and p is still the current node. In (2b), a new u is created at q, which is a neighbor of p but not of any other node marked u, and the new current node is q. Thus the path has been

lengthened, but still does not touch or cross itself. Finally, in (2c), the u at p is changed to v and the new current node is a neighbor of p marked u (by induction hypothesis, there is only one such neighbor, namely n_{k-1}); here the path has been shortened. ▮

COROLLARY. *When NT stops, p is the initial node, and all u's have been turned into v's.*

Proof. At the last entry into (2), when (3b) holds, we must have $k = 1$ (since p has no neighbor marked u). Thus there are no u's left; but u's can only turn into v's. ▮

PROPOSITION A7. *When NT stops, every node of ω has been marked v.*

Proof. When NT stops, there can be no nodes marked x, y, z, or u, and there is at least one node marked v. Suppose there were an unmarked node r; since ω is connected, there must thus be such an r that has v's as neighbors. Let s be the neighbor of r that *last* got marked v. By (3), just before s was marked v it had no unmarked neighbors; thus at that stage, r must have been marked y. But by (2a), any node marked y must be a neighbor of the current node (here: s) and must also have another neighbor (call it t) marked u. Thus when s is marked v, node r still has a neighbor t marked u, and this u must eventually get changed to v by NT; this contradicts the assumption that s is the last neighbor of r to get marked v. ▮

The foregoing Propositions show that NT, starting at node n on any unmarked web ω, will stop at n with every node of ω marked v. It follows that NT can be modified, by interchanging the roles of nodes marked v and unmarked nodes, so that when it starts at n on a web with every node marked v, it stops at n with every node unmarked (and the original node labels preserved intact throughout). Thus NT, together with this "reversal" of it, constitutes our desired universal node traverser.

RECEIVED: April 4, 1974; REVISED: January 8, 1975

BIBLIOGRAPHY

ABE, N., MIZUMOTO, M., TOYODA, J., AND TANAKA, K. (1973), Web grammars and several graphs, *J. Comput. Sys. Sci.* **7**, 37.

BEHZAD, M., AND CHARTRAND, G. (1971), "Introduction to the Theory of Graphs," Allyn and Bacon, Boston, Mass.

EARLEY, J. (1971), Toward an understanding of data structures, *Comm. ACM* **14**, 617.

EHRIG, H., PFENDER, M., AND SCHNEIDER, H. J. (1973), Graph grammars: An algebraic approach, *Proc. 14th SWAT*, 167–179.

FEDER, J. (1971), Plex languages, *Inform. Sci.* **3**, 225.

FISHER, G. A., AND RANEY, G. N. (1969), On the representation of formal languages using automata on networks, *Proc. 10th SWAT*, 157–165.

HARARY, F. (1969), "Graph Theory," Addison–Wesley, Reading, Mass.

HOPCROFT, J. E., AND ULLMAN, J. D. (1969), "Formal Languages and Their Relation to Automata," Addison–Wesley, Reading, Mass.

IBARRA, O. (1972), A note concerning nondeterministic tape complexities, *JACM* **19**, 608.

MINSKY, M. (1967), "Computation: Finite and Infinite Machines," Prentice–Hall, Englewood Cliffs, New Jersey.

MONTANARI, U. G. (1970), Separable graphs, planar graphs, and web grammars, *Inform. Contr.* **16**, 243.

MYLOPOULOS, J. (1972), On the relation of graph grammars and graph automata, *Proc. 13th SWAT*, 108–120.

PAVLIDIS, T. (1972), Linear and context-free graph grammars, *JACM* **19**, 11.

PFALTZ, J. L. (1972), Web grammars and picture description, *Computer Graphics and Image Processing* **1**, 193.

PFALTZ, J. L., AND ROSENFELD, A. (1969), Web grammar, *in* "Proceedings of the International Joint Conference on Artificial Intelligence" (D. E. Walker and L. M. Norton, Eds.), pp. 609–619, Washington, D.C.

PRATT, T. W. (1971), Pair grammars, graph languages, and string-to-graph translations, *J. Comput. Sys. Sci.* **5**, 560.

ROSENFELD, A., AND MILGRAM, D. L. (1972), Web automata and web grammars, *in* "Machine Intelligence 7" (B. Meltzer and E. Michie, Eds.), pp. 307–324, Edinburgh University Press, Edinburgh.

ROSENFELD, A., AND STRONG, J. P. (1971), A grammar for maps, *in* "Software Engineering" (J. T. Tou, Ed.), Vol. 2, pp. 227–239, Academic Press, New York.

ROSENSTIEHL, P., FIKSEL, J. R., AND HOLLIGER, A. (1972), Intelligent graphs: Networks of finite automata capable of solving graph problems, *in* "Graph Theory and Computing" (R. C. Read, Ed.), pp. 219–265, Academic Press, New York.

RUTLEDGE, J. D. (1970), Program schemata as automata, *Proc. 11th SWAT*, 7–24.

SAVITCH, W. J. (1972), Maze recognizing automata, *Proc. 4th ACM Symp. Theory Comput.*, 151–156.

SCHNEIDER, H. J. (1970), Chomsky-like systems for partially ordered symbol sets, *Erlangen Technical Report V3, N3.*

SCHWEBEL, J. C. (1972), A graph structure model for structural inference, *in* "Graphic Languages" (F. Nake and A. Rosenfeld, Eds.), pp. 195–209, North–Holland, Amsterdam.

SHAH, A. N., MILGRAM, D. L., AND ROSENFELD, A. (1973), Parallel web automata, *Univ. Maryland Tech. Rep. TR*-231, College Park.

SHANK, H. S. (1971a), Graph property recognition machines, *Math. Syst. Theory* **5**, 45.

SHANK, H. S. (1971b), Records of Turing machines, *Math. Sys. Theory* **5**, 50.

SHAW, A. C. (1972), Picture graphs, grammars, and parsing, *in* "Frontiers of Pattern Recognition" (S. Watanabe, Ed.), pp. 491–510, Academic Press, New York.

Printed in Belgium